Distributing NaturalX Applications

On Windows, UNIX and OS/390 platforms, NaturalX applications can be distributed using DCOM. Most information in this section is of a general nature. The section Configuration Examples supplies platform-specific information.

This section covers the following topics:

- General
- Globally Unique Identifiers (GUIDs)
- NaturalX Servers
- Activation Policies
- Registration
- DCOMPARM System Command (OS/390 Only)
- Type Information
- Configuration Overview
- Sample Application

General

Using NaturalX, you can make Natural classes and their services available to local and remote clients, thus creating distributed applications. Local clients are processes that run on the same machine as a given NaturalX server, and remote clients are processes that run on a different machine.

In order to distribute applications, a widely-used distributed object technology is used - the Microsoft Distributed Component Object Model (DCOM). When you register a Natural class to DCOM, its interfaces are presented to clients in a quasi-standardized fashion as dynamic COM interfaces, which are also known as dispatch interfaces. These interfaces can be easily addressed by many programming languages including Visual Basic, Java, C++ and, of course, Natural.

There are several points that must be taken into consideration when organizing the distribution of a NaturalX application. Each of these points is discussed in more detail in this chapter.

- Determine whether each class should be internal, external or local (see the section Internal, External and Local Classes).
- Globally unique IDs (GUIDs) must be assigned to the internal and external classes and their interfaces in order to be able to address them uniquely in the network (see the section Globally Unique Idenitfiers (GUIDs).
- You can define the activation policy for each class in order to control the conditions under which DCOM activates it (see section Activation Policies).
- In order to organize classes to applications, you can define NaturalX servers and assign the classes to them (see the section NaturalX Servers).
- Classes must be registered to make them known to DCOM (see section Registration).
- You can configure an application in order to further control its behavior (see the sections Configuration Overview and Configuration Examples).

Internal, External and Local Classes

It is important to distinguish between classes for internal use, classes for external use and those for local use only.

Internal Classes

The most important feature of internal classes is that their objects (instances) can only be created in the local client process.

Internal classes have the following features:

- Access to client session-dependent resources such as files and system variables.
- Can run within the client transaction.
- Can be debugged using the Natural Debugger.

External Classes

Windows 98/NT/2000 and UNIX

An external class can be created in the client process provided that the client process is simultaneously a server for the class. In addition, an external class can be debugged with the Natural Debugger (remote debugging).

OS/390

The most important feature of external classes is that their objects (instances) can *not* be created in the local client process.

All Platforms

External classes have the following features on all platforms:

- No access to client session-dependent resources such as stacks, files and system variables.
- Do not run within the client transaction.
- Can be used by remote nodes.
- Can be used by various clients using a variety of languages such as Natural, Java, Visual Basic, C/C++, etc.

Local Classes

Local classes are classes which are executed in local execution mode. Natural executes a class locally (within the Natural session) if it is either not registered or if DCOM is not available.

Local classes have the following features:

- Can be used even if DCOM is not available.
- Need not be registered with DCOM.
- Cannot be used from outside the client process.

Globally Unique Identifiers - GUIDs

DCOM uses global unique identifiers (GUIDs) - 128-bit integers that are virtually guaranteed to be unique throughout the world - to identify every interface and every class. This helps to ensure that server components can be located and to prevent clients connecting to an object accidentally.

If a class is to be registered to DCOM, every interface defined in a Natural class and the class itself must be associated with such a globally unique ID.

Once a globally unique ID has been assigned to an interface or a class, the ID must never be changed.

Using the Class Builder

On Windows NT and Windows 2000, Natural provides the Class Builder as the tool to develop Natural classes.The Class Builder automatically assigns a GUID to every class and interface.

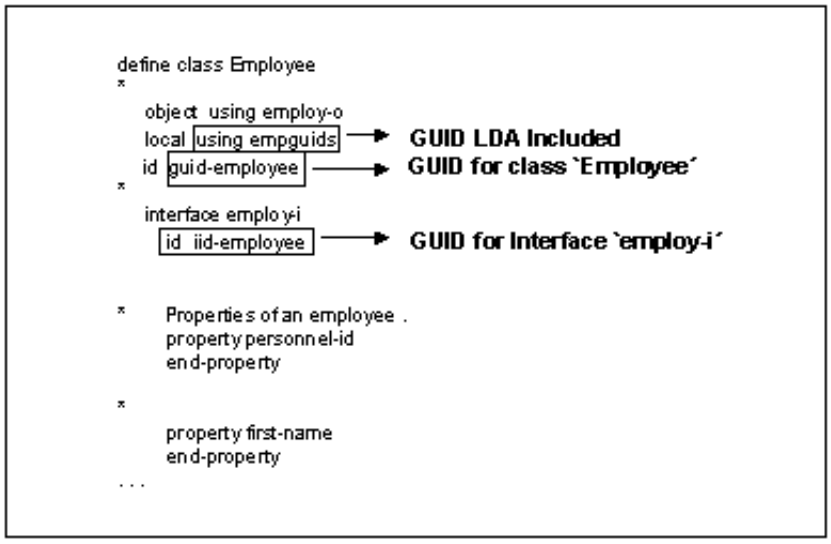
Using the Data Area Editor

On OS/390 and UNIX, you must use the Data Area Editor to create GUIDs. GUIDs are alphanumeric constants of type A36 in Natural. If you use this approach, it is suggested that, for ease of administration, you use one Local Data Area to store all the GUIDs for one project or application.

Note:
GUID generation is a functionality of COM and on the mainframe available in Batch and TSO only.

1. Create an LDA and insert one or more GUIDs, each provided with a symbolic name.
   For further information, see your Natural User's Guide.
   These symbolically-named constants are inserted into the data area and are initialized with an internally-generated globally unique ID.
2. Include the GUID LDA at the top of the class and use these named constants in your class and interface definition.

Example Including an LDA containing a GUID Definition in a Class Definition



NaturalX Servers

This section covers the following topics:

- COM Classes and Servers
- NaturalX Classes and Servers
- NaturalX Servers and Natural Sessions under OS/390
- NaturalX Servers and Natural Sessions under Windows 98/NT/2000 and UNIX
- The Role of the Server ID
- Organizing Server IDs

COM Classes and Servers

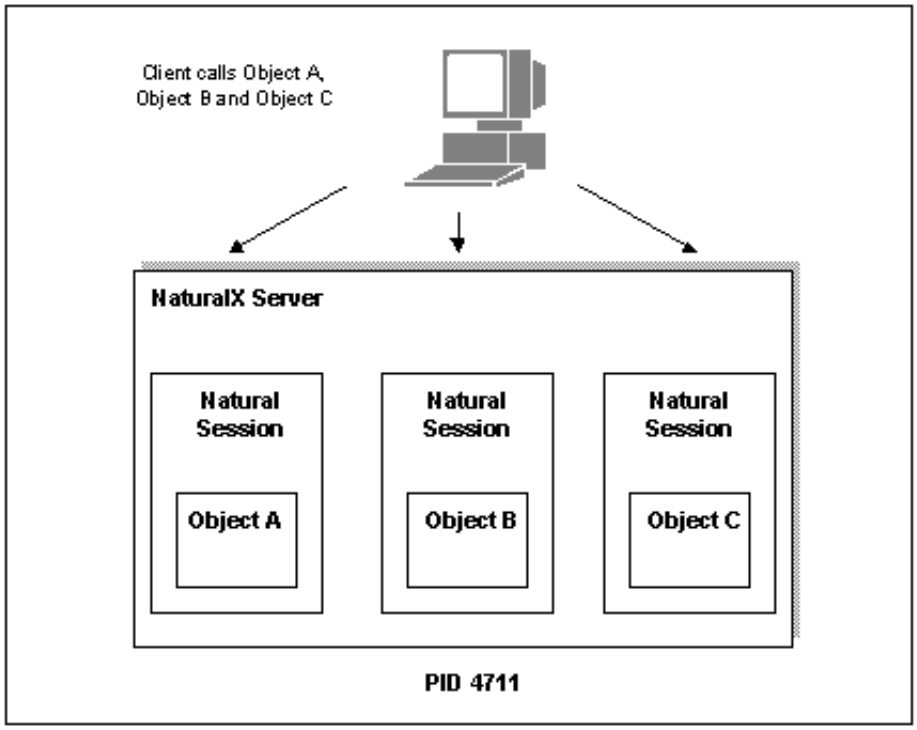
Each COM class must be hosted by a server process. The server process has a number of administrative and technical responsibilities, such as making the class and its interfaces available to DCOM and maintaining the memory occupied by the objects created. Whenever a client requests a new object of a certain class, DCOM checks whether the corresponding server process is already running. If this is not the case, DCOM launches it and passes the request to the server. When the server starts up, it makes its classes available to DCOM. While the server is running, it executes client requests for creation and deletion of objects and execution of methods. When the last object maintained by a server is deleted, the server shuts down automatically. For more detailed information about DCOM classes and servers, please refer to the Microsoft DCOM specification.

NaturalX Classes and Servers

Classes implemented with Natural can be made accessible as DCOM classes. But with Natural, it is not necessary to implement DCOM servers to host the classes. Instead, NaturalX itself performs the tasks of a DCOM server. NaturalX acts as a generic DCOM server for all classes written in Natural. The task that remains for a Natural class developer is just to implement the classes and to assign them to a NaturalX server.

NaturalX Servers and Natural Sessions under OS/390

Under OS/390, a Natural DCOM server process can manage several Natural sessions in parallel. This means that objects from different clients which request the same server are all hosted by the same server process (region). Each client exclusively owns its own Natural session.



Starting NaturalX Servers

EntireX DCOM launches a NaturalX server automatically if a client requests a server which is not currently active. You can also start a NaturalX server manually under OS/390 UNIX services, under TSO or in batch by executing the load module "naturalx" as follows:

Example for OS/390 UNIX Shell

```
naturalx "fuser=(10,32) profile=(dcomqa, 10,930) DCOM=(SERVID=qatest01)"
```

Example for TSO

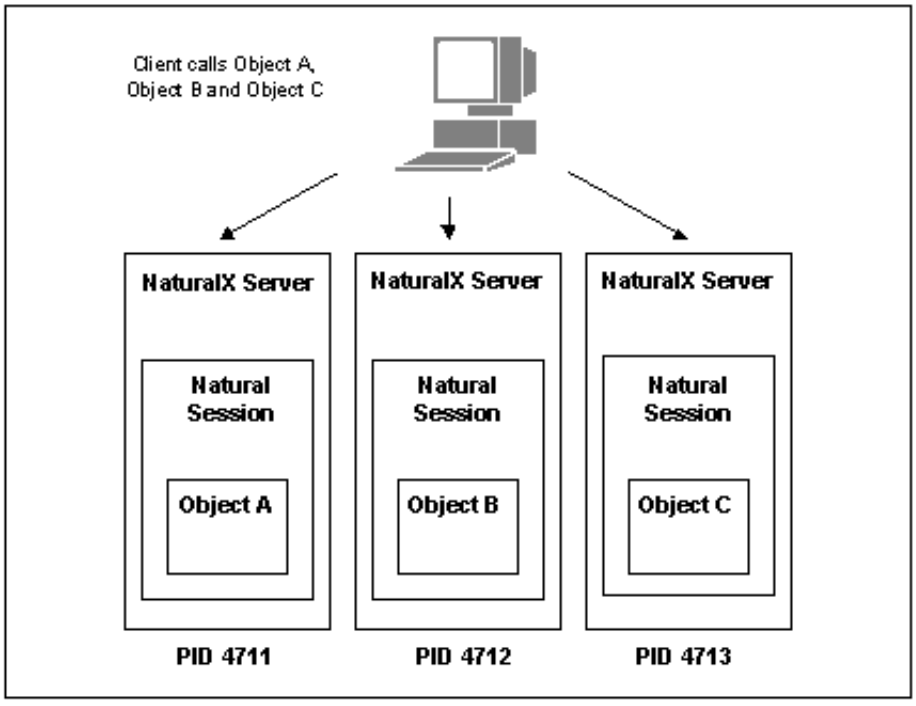
```
 PROC 0
   CONTROL NOFLUSH ASIS LIST CONLIST
   ALLOC FILE(NATXENV) +
         PATH('/u/nat/natxenv') PATHOPTS(ORDONLY)
   ALLOC FILE(CMPRINT)  DA(*)
   ALLOC FILE(CMSYNIN)  DA(*)
   ALLOC FILE(SYSOUT)   DA(*)
   ISPEXEC LIBDEF ISPLLIB DATASET ID('PRD.NXX111.LOAD' -
                                     'PRD.NAT311.LOAD')
   CALL 'PRD.NXX111.LOAD(NATURALX)' +
     'profile=(dcomqa,10,930),dcom=(servid=qatest01)'
   END
```


Example for Batch

```
//NXXSVR JOB CLASS=K,MSGCLASS=X
  //NXX      EXEC PGM=NATURALX,REGION=3200K,
  // PARM=('profile=(dcomqa,10,930),',
  // 'dcom=(servid=qatest01)')
  //STEPLIB  DD DISP=SHR,DSN=PRD.NXX111.LOAD
  //         DD DISP=SHR,DSN=PRD.NAT311.LOAD
  //SYSUDUMP DD   SYSOUT=X
  //NATXENV  DD   PATH='/u/nat/natxenv',PATHOPTS=(ORDONLY)
  //CMPRINT  DD   SYSOUT=X
  /*
```

NaturalX Servers and Natural Sessions under Windows 98/NT/2000 and UNIX

Under Windows 98/NT/2000 and UNIX, each Natural session runs in its own exclusive NaturalX server process.



The Role of the Server ID

One of the tasks of a DCOM server is to make its classes available to DCOM during startup. But since NaturalX acts as a generic DCOM server, it has no built-in knowledge about the classes it shall provide. Instead, it finds the list of these classes in the system registry under the key of its server ID. The server ID is a Natural-owned key in the system registry, keeping together all classes that belong to a given NaturalX server. It is an arbitrary alphanumeric string of 32 characters which does not contain blanks and which is not case sensitive.

How does a NaturalX server know under which server ID it is running? The server ID is defined with the Natural parameter COMSERVERID=*serverid* (for OS/390, DCOM=(SERVID=*serverid*)). This parameter is either passed to a NaturalX server as a dynamic parameter on the command line, or it is defined in the Natural parameter module.

How are classes assigned to server IDs? Assume Natural has been started with a certain server ID. Then every class that a user registers during this Natural session is entered into the system registry under the current server ID.

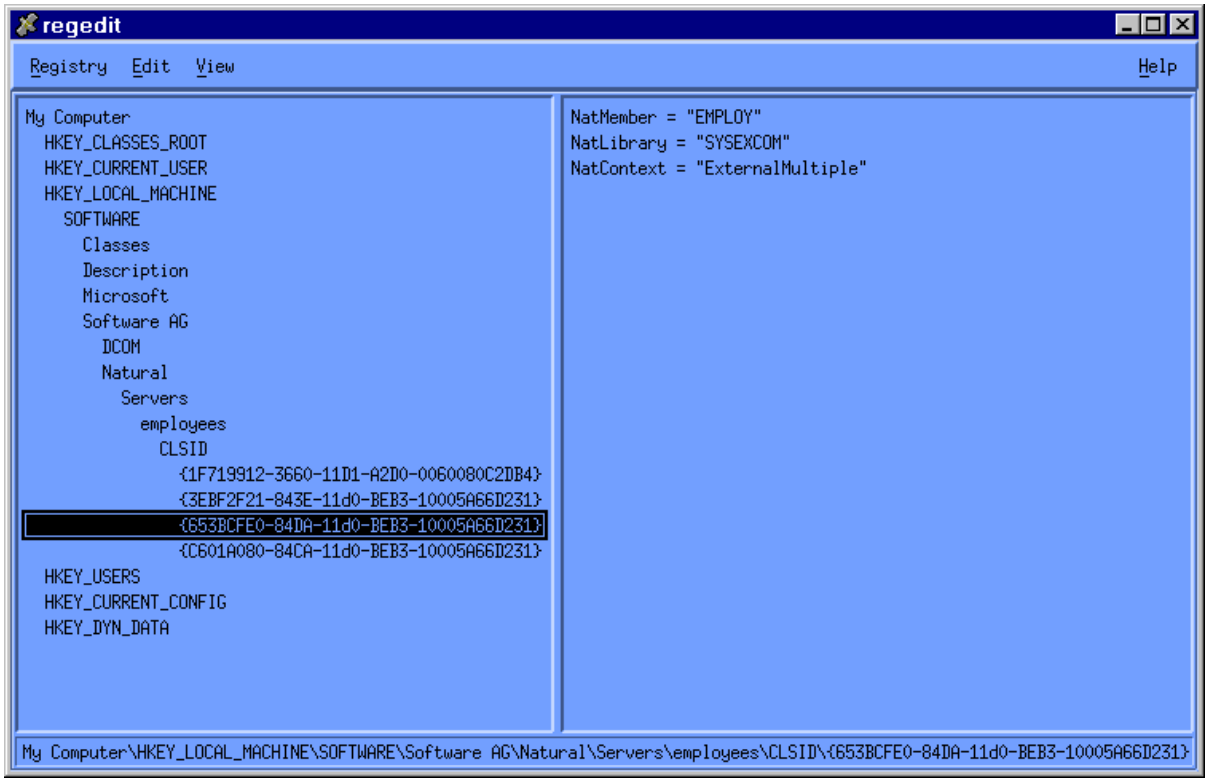
Server IDs provide a means of grouping classes created in Natural and assigning them to different NaturalX server processes. The use of server IDs is, however, not compulsory: if Natural is started without a server ID, all Natural classes are registered under the predefined server ID "Default".

Example

Consider the example *Employees* application consisting of the classes *DepartmentList*, *EmployeesList* and *Employee* (this application is contained in the example library SYSEXCOM). These three classes are to be hosted by a NaturalX server called *Employees*.

1. Start Natural with the desired server ID.
2. Logon to the library SYSEXCOM.
   LOGON SYSEXCOM
3. Register the classes with the REGISTER command on the Natural command line
   REGISTER *

Note:
On the mainframe, the REGISTER command is only available under TSO and in batch.
For further information, see the section The REGISTER Command.
The three classes are now registered under the server ID *Employees.* The followingexample shows the Registry Editor under UNIX.



Whenever an object of one of these classes is requested, DCOM will start a NaturalX server process with the server ID *Employees*, which will then provide the classes.

Organizing Server IDs

The server ID represents the set of all classes that are made available to DCOM when the corresponding NaturalX server is started. It is recommended that you group under one server ID those classes that form an application from the business point of view, or that otherwise belong together logically. Similarly, classes that are never used in the same context should be registered under different server IDs. Another criterion for the assignment of classes to server IDs is security (see the section Security). From this aspect, it makes sense to group under the same server ID those classes for which common authorizations will be defined.

Activation Policies

This section covers the following topics:

- Activation Policies Under Windows 98/NT/2000 and UNIX
- Activation Policies Under OS/390
- Setting Activation Policies
- When to use which Activation Policy

Activation Policies Under Windows 98/NT/2000 and UNIX

If a client makes a request to create an object of a certain class, it is DCOM's task to start a server process that provides the class and to direct the request to this process. For Natural classes, the responsible server process is a NaturalX server. DCOM recognizes different options that control when a new server process is started or when an object is created in a server process that is already running. For further information, see the section Registration. While registering a Natural class with the REGISTER command, you can control which activation options DCOM shall use for this class. NaturalX combines the different options supported by DCOM in the form of the following three activation policies:

- *ExternalMultiple*
  If a Natural class is registered with the activation policy *ExternalMultiple*, and a client requests an object of that class, DCOM tries first to create the requested object in the current process. Remember that the client itself might at the same time be a NaturalX server and might provide the class itself. If the current process is not a server for the class, DCOM starts a new NaturalX server process and creates the object in that process. If a second object of the same class is created later, this object is also created in that server process. This means that the same server process can contain several objects of the class.
- *ExternalSingle*
  If a Natural class is registered with the activation policy *ExternalSingle*, DCOM starts a new NaturalX server process each time an object of this class is created. One server process can contain only one object of the class.
- *InternalMultiple*
  If a Natural class is registered with the activation policy *InternalMultiple*, DCOM always creates objects of this class in the current process. The same server process can contain several objects of the class.

The default activation policy is *ExternalMultiple*. This default is defined with the Natural parameter ACTPOLICY and can be changed with the NATPARM utility.

Activation Policies Under OS/390

Activation policies are used to decide whether a class requires an exclusive Natural session or whether it can share one with other objects. As far as DCOM is concerned, a NaturalX server is responsible for all the objects of those classes which are registered with the same server ID. That is, for any one server ID, only one NaturalX server process is active.

Activation policy is evaluated by the NaturalX server. The following two sets of options are supported:

- External/Internal
  Determines whether an object is created in the Natural client session requesting it (internal), or in a different Natural session (external).
- Single/Multiple
  Determines whether an object requires a Natural session for its exclusive use (single) or not (multiple).

NaturalX combines the above options to form the following three activation policies with which classes can be registered:

- *InternalMultiple*
  An object is created within the current Natural session.
- *ExternalMultiple*
  An object is created in a different Natural session which may accomodate multiple objects. Because it is not possible to distinguish between multiple client applications running under the same user ID on the same node, all objects created by the same user on the same node are hosted by the same single Natural session. This is valid even if the objects are created from within different applications.
- *ExternalSingle*
  Each object occupies a separate Natural session for its exclusive use.

Note:
Even though objects registered as ExternalMultiple can coexist with other objects in a Natural session, the multiple objects requested by one client are collected within one session. This means that objects form different clients can not interfere with each other.

Setting Activation Policies

The activation policy of a class can be set in three different ways, in the following order of precedence:

- Explicity as part of the REGISTER command
- In the DEFINE CLASS statement
- With the profile parameter ACTPOLICY=*activation-policy* (for OS/390, DCOM=(ACTPOL=*activation-policy*))

When to use which Activation Policy

Non-trivial DCOM applications will mostly deal with *persistent* objects, i.e. objects stored in databases. For such applications, some considerations concerning database access, transaction handling and user isolation must be made. Consider the following scenario: clients A and B both create an object of a class that is provided by a certain NaturalX server process. Assume that the NaturalX server uses a database to load and store its objects. If both clients were served by the same server process, they would appear to the database as one single user. This would have the consequence that a transaction started by a method call from Client A can be committed or backed out by a method call from Client B. Such interferences are obviously to be avoided.

There are two approaches to avoiding this interference: either the clients do not use persistent objects, or each of them is served by its own NaturalX server process. Both approaches have their advantages in different situations; for a class or application that does not access databases or other shared resources, it is useful to serve several clients with a single server process. For classes that access databases or other shared resources, it is necessary to isolate different clients in different server processes. Hence both approaches should be possible. Activation policies give an administrator the means to control the activation behavior for each class at registration time.

Example

This example illustrates how the various activation policies can be used. Let us consider parts of an imaginary travel agency application. The application contains the business classes *Trip*, *Skipper* and *RoutePlanner*. The *Trip* class represents a sailing trip to be planned, the *Skipper* class represents the skippers available to lead the trips. *RoutePlanner* is a class that determines an optimal route for a trip. Assume that the *Trip* and *Skipper* classes use a database to read and store their objects. The *RoutePlanner* class just performs some calculations on a given *Trip* object and does not use a database.

Since some of the business classes use transactional access to a database, and a transaction might span several method calls, each active client needs to be served with its own NaturalX server process. This can be done by defining an additional class *SagTours*, which represents an application session. This class can be used, for example, to keep general information about the session status, but the main task will be to create business objects on behalf of a client.

Class SagTours

```
* Represents a SagTours application session.
  *
  define class SagTours
    local using tour-ids
    id clsid-sagtours
  *
    interface Create  /* Used to create application objects. */
      id iid-sagtours-create
  *
      method newTrip  /* Creates a new Trip object. */
        is trip-n
        parameter
        1 trip handle of object by value result
      end-method
```



```
      method newSkipper   /* Creates a new Skipper object. */
        is skip-n
        parameter
        1 skipper handle of object by value result
      end-method
  *
  end-interface
  *
  end-class
  end
```

This class will be registered as *ExternalSingle*. This means that each creation of a *SagTours* object starts a NaturalX server process for the client that requested the object. A client will create a *SagTours* object only once and will use its methods later to create the business objects it needs. In order to create a *Trip* object, the client will call the method *newTrip*, which is implemented as follows:

Method newTrip

```
  * This method creates a new Trip object.
  *
  define data parameter
  1 trip handle of object by value result
  end-define
  *
  create object trip of class "Trip"
  *
  end
```

The *Trip* class itself will be registered as *InternalMultiple*. This ensures that the *Trip* objects created by the method *newTrip* are created in the NaturalX server process just started for this client.

Now let us look at the class *RoutePlanner*.

Class RoutePlanner

```
* Plans optimal routes for sailing trips.
  *
  define class RoutePlanner
    local using tour-ids
    id clsid-planner
  *
    interface routing
      id iid-planner-routing
  *
      method plan   /* Plans a sailing trip. */
        is plan-n
        parameter
        1 trip handle of object by value
      end-method
  *
    end-interface
  *
  end-class
  end
```

Method plan

```
* This method plans a sailing trip.
*
define data parameter
1 trip handle of object by value
end-define
*
* Perform some operations on the given Trip object.
*
end
```

This class can be registered as *ExternalMultiple*. In this case, all *RoutePlanner* objects created by different clients would be created in the same NaturalX server process. This does not do any harm if the methods of this class do not access databases, or if each database transaction is fully contained in a method (i.e. if each method subprogram ends with either a BACKOUT TRANSACTION statement or an END TRANSACTION statement).

Now let us look at a sample client program:

Sample Client Program

```
define data local
 sagTours handle of object
 trip handle of object
 planner handle of object
end-define
*
* Start the application session.
create object sagTours of "SagTours"
*
* Create a Trip object.
send "newTrip" to sagTours return trip
* Create a RoutePlanner object.
create object planner of "RoutePlanner"
* Plan the trip.
send "plan" to planner with trip
*
end
```

The client first creates a *SagTours* object. This starts a new NaturalX server process exclusively for this client. The client then uses the *SagTours* object to create a *Trip* object in the context of this application session. Note that the client creates the *RoutePlanner* object directly. This is possible because the class is registered as *ExternalMultiple*, but it is not necessary: the *SagTours* class could also provide a method for the creation of *RoutePlanner* objects. Afterwards it lets the business objects do their jobs. The objects are automatically released at program end. The deletion of the *SagTours* object causes the NaturalX server to shut down.

Note:
This example shows only the NaturalX techniques needed to illustrate the usage of activation policies. A real-world application would require a lot more. The classes would use object data areas and they would surely have globally unique IDs assigned. Also parameter data areas would be used instead of inline parameter declarations.

Registration

If a class is to be made accessible to DCOM clients, it is necessary to add some information about the class to the system registry. DCOM clients will mostly address a class with a meaningful name, the so-called programmatic identifier (ProgID) as in the following example:

```
CREATE OBJECT #O1 OF CLASS "Employee"
```

For a Natural class, the class name defined in the DEFINE CLASS statement is written into the registry as a ProgID.

System registry entries map this ProgID to the globally unique ID (GUID) of the class, allowing DCOM to uniquely locate all information about the class. Further information that is stored in the registry includes the path and name of the responsible DCOM server, the path and name of the type library, and interface information.

This section covers the following topics:

- Registration with Natural
- Automatic Registration
- Manual Registration
- Registration Files and Type Library
- Client Registration
- Registration Hints

Registration with Natural

Natural classes can be registered (or unregistered) manually with the system command REGISTER(orUNREGISTER), automatically after the class is stowed (or deleted), or by running the .reg files which are generated every time a class is registered.

In order to register classes, you must have the rights to modify the system registry and your system environment must be able to use COM.

It is usually not advisable to change the Natural entries in the system registry directly in the registry editor because this can lead to inconsistent registry entries.

A class is always registered for the server ID under which Natural was started.

Note:
On the mainframe, the REGISTER command is only available under TSO and in batch.

Automatic Registration

If the profile parameter AUTOREGISTER(for OS/390 DCOM=(AUTOREG=*value*)) is set to ON, a Natural class is automatically registered when it is stowed (cataloged), and unregistered automatically when it is deleted. This means that the user can test the class directly after stowing it.

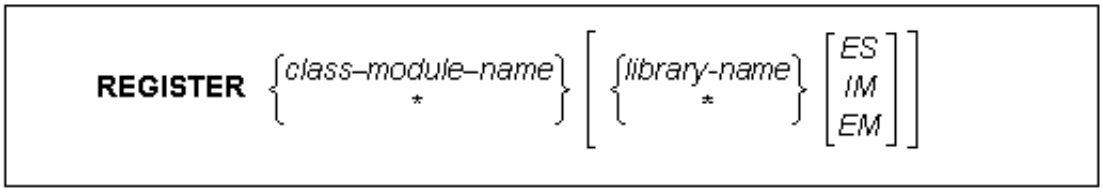
Automatic registration uses the activation policy setting defined in the WITH ACTIVATION POLICY clause of the DEFINE CLASS statement of the class. If this clause is not specified, the setting from the profile parameter ACTPOLICY=*activation-policy* (for OS/390, DCOM=(ACTPOL=*activation-policy*)) is used.

If automatic registration is set and a class is stowed (cataloged), the class is unregistered before it is stowed and registered after the stow has finished so that all old registry entries are removed.

Manual Registration

The REGISTER Command

The system command REGISTER is used to register Natural classes. They are registered for the server ID under which Natural was started.



Note:
On the mainframe, the REGISTER command is only available under TSO and in batch.

class-module-name

This defines which class or classes are to be registered by specifying the appropriate Natural object module name.

library-name

This defines which library or libraries are to be searched for the class or classes.

ES IM EM

This defines the activation policy which is registered for the class or classes.

You can set one of the following parameters:

Parameter	Description
ES	Sets activation policy *ExternalSingle*
IS	Sets activation policy *InternalSingle*
EM	Sets activation policy *ExternalMultiple*

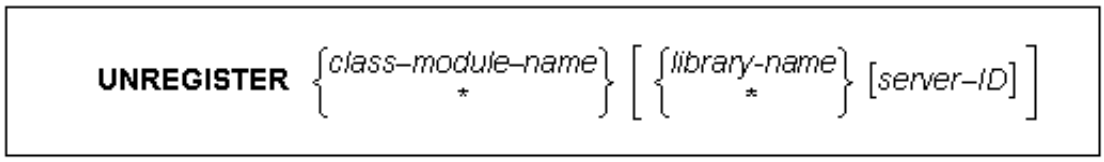
The following table shows which classes will be registered for all possible class/library combinations:

Class Module Name Specification	**Library Name Specification**		
library-name	*	-	
class-module-name	class with class module name *class-module-name* of library *library-name*	all classes with the class module name *class-module-name* which are found in the current step libraries	class with class module name *class-module-name*
*	all classes which are found in the library *library-name* are registered	all classes which are found in the current step libraries are registered	all classes of the current logon library are registered

If this parameter is not specified in the REGISTER command or the DEFINE CLASS statement, the default activation policy defined in NATPARM is used.

The UNREGISTER Command

The system command UNREGISTER is used to unregister Natural classes.



class-module-name

This defines which class or classes are to be unregistered by specifying the appropriate Natural object module name.

library-name

This defines the library or libraries which are to be searched for the class or classes.

server-ID

This defines the server ID of the class or classes.

The following table shows which classes will be unregistered for all possible class/library/server ID combinations:

Class Name Specification	**Library Name /Server ID Combination**				
	- -	***library-name* -**	***library-name server-ID***	*** -**	*** *server-ID***
class-module-name	class with *class-module-name* in the current logon library if it is registered for the current server ID	class with *class-modulename* of library *library-name* if it is registered for the current server ID	class with *class-module-name* of library *library-name* if it is registered for the server *server-ID*	all classes with *class-module-name* found in the current step libraries if they are registered for the current server ID	all classes with the name *class-module-name* found in the current step libraries which are registered for the server *server-ID*
*	all classes of the current logon library which are registered for the current server ID	all classes found in the library *library-name* which are registered for the current server ID	all classes found in the library *library-name* which are registered for the server *server-ID*	all classes found in the current step libraries which are registered for the current server ID	all classes found in the current step libraries which are registered for the server *server-ID*

A REGISTER or UNREGISTER system command will return an error message if *class-module-name* or *class-module-name* and *library-name* are specified but either the class or library is not found. If only an asterisk (*) is given in the REGISTER or UNREGISTER system command, no error message is returned if no class has been registered or unregistered.

If a class without class GUIDs or interface GUIDs is specified in the REGISTER system command, an error message will be returned. Such a class can only be used in the local Natural session.

Registration Files and Type Library

Registration files (.reg files) enter information in the system registry when they are executed.

Natural will automatically create registration files for the server and the client side when a class is registered.

The server .reg file contains the same information that was entered in the system registry and the client .reg file contains all information which is generated for the client side. When a class is unregistered, the .reg files will be deleted. If a .reg file is not to be deleted with the unregistration, the file has to be renamed before unregistering the class because Natural deletes only files with the default .reg file names.

The .reg files will be named <classmodule_name>_S.reg (for the server) and <classmodule_name>_C.reg (for the client) and, to activate a different version, <classmodule_name>_V.reg.

A type library is created automatically when a class is registered, and it is deleted when a class is unregistered. A reference to the type library is also entered in the registry.

The default type library name is <classmodule_name>.tlb. A new name will be generated if a type library with this name exists already.

The registration files and the type library are stored in the Natural *etc*-directory as follows:

$NATDIR/$NATVERS/etc/<serverid>/<classname>/v<version-number>

Example

The files for version one of a class MY.TEST.CLASS registered for the server ID SERVER01 are located as follows:

$NATDIR/$NATVERS/etc/SERVER01/MY.TEST.CLASS/v1

Client Registration

Natural does not enter the registration information for the clients automatically in the system registry, but creates a registration file for the client. The client registration file contains an entry (RemoteServerName) that tells DCOM on which machine the DCOM server class can be found. This entry is not filled from Natural. It can be entered in either of two ways:

1. The RemoteServerName can be entered in the registration files. In this case the line

   ```
   "RemoteServerName"=
   ```

   has to be changed to

   ```
   "RemoteServerName"="<server_machine_name>"
   ```

   After this, the registration file has to be executed on the client machine.

2. The registration file is executed first, and then the RemoteServerName is changed using the DCOMCNFG tool or the Registry Editor (see the section Configuration Examples).

Registration Hints

The following points should be taken into account when registering and unregistering classes:

- The class GUID should never be changed for an existing class: Natural displays an error message if a class that is already found in the registry is registered again with another GUID. The old class must first be unregistered in this case.
- The same class should never be registered for more than one server ID: there is a one-to-one relationship between the server ID and the AppID, and a class has only one AppID defined, which means that a registration for a second server ID overwrites the AppID. Furthermore, if the class is unregistered for one server ID, all entries of the class are removed without checking whether it is registered for a second server.
- Except for client registration, you should always use the Natural system commands REGISTER and UNREGISTER to change registry entries for a class because they remove redundant registry entries.
  For example, if a client class has been registered for server1 and a server registration file with a registration of the same class for server2 is run, the AppID key of the class is changed and all references to the old AppID key are lost. So this old AppID key can never be deleted. When a class is registered with the system command REGISTER, a check is made to see whether the AppID has been changed, and the old AppID is removed if no other class needs it.
- If Natural is not available on the client machine and registry entries for a Natural class are to be removed from the system registry, you should do this with the registry editor. If Natural is available on the client machine, it is easier to register the class first with the Natural system command REGISTER and unregister it afterwards with the system command UNREGISTER.
- The registration information for a class is taken from the catalogued class object, so that it is not possible to register or unregister a class that is only available in source format.
- If you want to register classes during a Natural session, the session must be started with the parameters PARM and COMSERVERID=*server-ID* (for OS/390, DCOM=(SERVID=*server-ID*)) only as shown below. This is because only these two parameters are stored in the registry key "LocalServer32". If a class is tested with other parameter settings, there is no guarantee that it will run later when it is started from a DCOM client.

  ```
  Windows 98/NT/2000:
    NATURAL.EXE PARM=COMPARM COMSERVERID=SERVER1
    UNIX:
    NATURAL PARM=COMPARM COMSERVERID=SERVER1

    OS/390:
    NATURAL DCOM=(SERVID=SERVER1)
  ```

- Session Parameters for NaturalX Servers (OS/390 Only): If DCOM launches the server, the session parameters defined in the system registry are used. Although the REGISTER command only sets the parameter DCOM=(SERVID=*server-id*), it is possible to maintain the server session parameters in the system registry using the Natural system command DCOMPARM. If you start the server manually from the shell, you can specify dynamic session parameters with the shell command. For example:

  ```
  naturalx "profile=myprofile fnat=(10,930) dcom=(serverid=employees)"
  ```

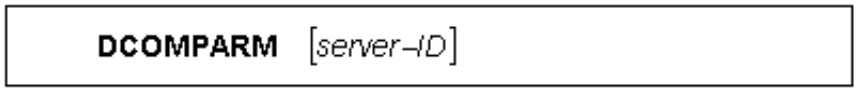
  If you start the server manually in batch/TSO, you can specify dynamic session parameters with the "naturalx" command or via the dataset name CMPRMIN.

- Usually only users with administrator rights can change the system registry. So if you receive an error when trying to register a class, check to see whether you have the rights required to change the registry.
- When a Natural class is registered, some additional information is entered in the registry that is only needed by Natural (not by DCOM). The information which is stored in the additional registry keys is the server ID (see section NaturalX Servers), the activation policy (see section Activation Policies) and the location (Natural class module name and library of class) of the class. This information is necessary, for example, if all classes of a specified server ID are to be unregistered or to make the served classes available when Natural is started.
- There is a one-to-one relationship between the server ID and the AppID (under HKEY_CLASSES_ROOT/AppID) of a class. When a class is registered for a new server ID, a new GUID - the

AppID, is generated and assigned to this server ID. The AppID is used by DCOM to group the DCOM classes. Security settings and (for client registrations) the remote machine name are defined for an AppID, i.e. all classes which belong to one AppID have the same security settings (see the sections Configuration Overview and Security).

DCOMPARM System Command - OS/390 Only

The system command DCOMPARM is used to display and modify the Natural parameters for a specified server ID in the system registry.



The parameters for the specified server ID are read from the system registry and are displayed in the ‘DCOM Parameters’ screen, where they can be modified. Make sure that no parameter is split at the end of an input line.

When you save your changes, NaturalX searches the parameter list for the parameter DCOM=(SERVERID=*serverID*). If the parameter is found, it is moved to the end of the list. If it is not found, it is created at the end of the parameter list. No other data validation takes place.

You can reread the parameters from the system registry for the specified server ID by issuing the READ command.

server-ID

Specify the server ID which was used for the Natural session in which the REGISTER command for the Natural class was executed.

Note:
You can find the server ID in the system registry.

Type Information

This section covers the following topics:

- Overview
- NaturalX and Type Information
- Using Type Information

Overview

Type information is a means to completely describe a class along with all of its interfaces, down to the names and types of the methods. It contains the necessary information about classes and their interfaces, for example, which interfaces exist on which classes, which member functions exist in those interfaces, and which argument those functions require.

This information is used by clients to find out details about a class and its methods, for example, by type-information browsers to present available objects, interfaces, methods and properties to an end user.

Another important area for using type information is the widely-used OLE automation technique which is also used by NaturalX.

There are several ways to store type information. A common way is generating the type information in type library (.TLB) files.

NaturalX and Type Information

Creating Type Information

For each Natural class, a type library file is created when the class is registered.

The type library is generated in the $NATDIR/$NATVERS/etc/<serverid>/<classname>/<version> directory and connected to the class via an entry in the registry.

The name of the class module is used, and the .tlb extension is appended unless the type library file name conflicts with an existing name. Then a number is attached to the class module name.

Using Type Information

Each interface defined in a Natural class is seen by clients as a dynamic interface (also called a dispatch interface). Each method of an interface is seen by clients under the name defined in the DEFINE METHOD statement.

The first interface in a Natural class is marked as the default dispatch interface.

The support of type information also makes it possible to define multiple interfaces with identical method/property names. The Natural client simply addresses the corresponding method by using the interface name (as defined in the Natural class) as the prefix of the method name, as shown in the following example:

```
CREATE OBJECT #O3 OF CLASS "DepartmentList"
  SEND "Iterate.PositionTo" TO #O3 WITH "C" RETURN #DEPT
```

Natural clients use type information to find out to which interface a method or property belongs.

Note:
Natural clients do not use type information at catalog time to perform syntax checks.

Data Type Conversions

Natural Types to OLE Types

In order to receive data from clients or to pass data to classes written in different programming languages, the Natural data types are converted to so-called OLE automation-compatible types. This table shows how clients see method parameters or properties of a Natural class. For example, if a Natural class has a method parameter or a property with the format A, this is seen by clients as VT_BSTR.

Natural Data Type	Automation-Compatible Type
A	VT_BSTR
B1	VT_UI1
B2	VT_UI2
B4	VT_UI4
B*n* (*n* != 1, 2, 4)	SAFEARRAY of VT_UI1
C	not supported
D	VT_DATE
F4	VT_R4
F8	VT_R8
I1	VT_I2
I2	VT_I2
I4	VT_I4
HANDLE OF GUI	not supported
HANDLE OF OBJECT	VT_DISPATCH
L	VT_BOOL
N15.4	VT_CY
Nn.m (n.m != 15.4)	VT_R8
P15.4	VT_CY
Pn.m (n.m != 15.4)	VT_R8
T	VT_DATE

An array of a given Natural type is mapped to a SAFEARRAY of the corresponding VT type.

There are, however, some special cases:

- A variable of type B*n*, where *n* is not 1, 2, 4 or an array of such types, is always mapped to a one-dimensional SAFEARRAY of VT_UI1. This is the way recommended by Microsoft to transport binary data through a dispatch interface.
- Control variables are not mapped. They have no meaning outside of Natural. Variables of type HANDLE OF GUI are also not mapped. There is no corresponding automation-compatible type. Therefore properties of the type Control variable or HANDLE OF GUI cannot be accessed by clients through COM/DCOM. Method parameters of these types should be marked as optional in the parameter data area, so that clients can omit the parameters when calling the method through COM/DCOM.

OLE Types to Natural Types

This table shows how parameters or properties of an external class can be addressed by Natural. For example, if an external class has a method parameter or property with format VT_R4, this parameter or property can be addressed in Natural as F4 or with a format that is MOVE-compatible to F4.

Automation -Compatible Type	Natural Data Type
VT_BOOL	L
VT_BSTR	A
VT_CY	P15.4
VT_DATE	T
VT_DISPATCH	HANDLE OF OBJECT
VT_UNKNOWN	HANDLE OF OBJECT
VT_I1	I1
VT_I2	I2
VT_I4	I4
VT_INT	I4
VT_R4	F4
VT_R8	F8
VT_U1	B1
VT_U2	B2
VT_U4	B4
VT_UINT	B4

A SAFEARRAY of up to three dimensions is converted into a Natural array with the same dimension count and the corresponding format. SAFEARRAYs with more than three dimensions cannot be used from within Natural.

There are, however, some special cases:

- A VT_BSTR maps either to a Natural Alpha variable or to a one-dimensional array of Natural Alpha variables. The additional dimension is used to store strings longer than 253 characters.
- A SAFEARRAY of VT_BSTRs maps either to an array of Natural Alpha variables with the same dimension count, or to an array with one more dimension. The additional dimension is used to store strings longer than 253 characters.
- A SAFEARRAY of VT_UI1 can be mapped to any array of Natural binaries that has a matching total size. This is because a SAFEARRAY of VT_UI1 is the way to transport binary data through a dispatch interface.

Configuration Overview

Once all classes of an application have been registered on the client and server machines, certain aspects of the application's behavior can be controlled and configured with system registry settings. This section summarizes the relevant registry entries and their meaning for NaturalX applications. For detailed background information about the registry keys and their administration, please refer to the specific DCOM registry documentation of the appropriate platform.

The registry keys relevant in this context are maintained with commonly-used tools like DCOMCNFG or the Registry Editor (REGEDIT). These tools present the registry keys in a different way. Therefore only the names of the registry keys are mentioned here. The section Configuration Examples describes how to set registry keys.

Note:
HKLM is the common short form of the registry key HKEY_LOCAL_MACHINE, where HKCR stands for HKEY_CLASSES_ROOT.

This section covers the following topics:

- Server Configuration - General Settings
- Server Configuration - Application-Specific Settings
- Client Configuration - General Settings
- Client Configuration - Application-Specific Settings

Server Configuration - General Settings

This section discusses general server configuration settings.

- The registry entry *HKLM\Software\Microsoft\OLE\EnableDCOM* must be set to 'Y' to enable access to the server machine via DCOM.
- If guests (users who do not have their own account on the server machine) are to be able to access applications on the server machine, the predefined account *Guest* must be enabled in the User Manager (Windows NT and Windows 2000 only).
- The registry entries *HKLM\Software\Microsoft\OLE\DefaultLaunchPermissions* and *HKLM\Software\Microsoft\OLE\DefaultAccessPermissions* define which users or groups are allowed or not allowed to launch DCOM applications and to access their classes. The authorizations defined here apply for all applications for which no application-specific settings are defined.
- The registry entry *HKLM\Software\Microsoft\OLE\LegacyAuthenticationLevel* controls the level of authentication that is performed for clients that access DCOM applications on this machine. If a NaturalX server is to be able to pass the client's user ID to Natural Security, the setting should be at least *Connect.* Choose *None* if no authentication is to take place. In this case, the NaturalX server does not retrieve the client's user ID. Instead it performs each request under the user ID under which it was launched. If this entry is defined differently on the client side and on the server side, the stricter setting applies.

- The registry entry *HKLM\Software\Microsoft\OLE\LegacyImpersonationLevel* controls how much information a server may retrieve about the client, or if it may even use this information to act in the role of the client against other servers. If a NaturalX server is to be able to pass the client's user ID to Natural Security, the setting should be at least *Identify*. The settings *Impersonate* or *Delegate* have the same effect for a NaturalX server. Choose *Anonymous*, if the server is not to be able to retrieve the client's user ID. In this case, the server performs each request under the user ID under which it was launched. If this entry is defined differently on the client side and on the server side, the stricter setting applies.

Server Configuration - Application-Specific Settings

The application-specific settings can be set up differently for each NaturalX application. But the question is where to apply these settings. It is important to remember that all classes registered under one NaturalX server ID form one application in the DCOM sense, and are thus assigned to one AppID key in the registry. This is why the application-specific settings are applied under the AppID key.

- The registry entries *HKCR\AppID\<APPID>\LaunchPermission* and *HKCR\AppID\<APPID>\AccessPermission* define which users or groups are allowed or not allowed to launch the DCOM application with the specified AppID and to access its classes.
- The registry entry *HKCR\AppID\<APPID>\RunAs* defines the user account this NaturalX server will run when it is launched by DCOM. There are three options:
  - *Interactive user:*
    The NaturalX server is started under the account of the user that is interactively logged in on the server machine. This is usually not desirable but can be useful for test reasons.
  - *Launching user:*
    The NaturalX server is started under the account of the client that creates the first object on this server (remember that the first request for an object forces DCOM to launch the server). This setting should be used if each client is to be served by its own server process. Obviously, the client must have permission to launch the server.
  - *This user:*
    The server is started under the account of a given user. This setting should be used if all clients are to be served by the same server process. The user entered here must have permission to launch the server.

Client Configuration - General Settings

This section discusses general client configuration settings.

- The registry key *HKLM\Software\Microsoft\OLE\LegacyAuthenticationLevel* controls the degree of authentication that is performed for clients running on this machine when they access DCOM applications. For a client that accesses a NaturalX server, a similar consideration to that inthe sectionServer Configuration - General Settingsapplies: only if it specifies at least *Connect*, will the NaturalX server be able to use its user ID against Natural Security. If this entry is defined differently on the client side and on the server side, the stricter setting applies.
- The registry key *HKLM\Software\Microsoft\OLE\LegacyImpersonationLevel* controls how much information a server may retrieve about the client, or if it may even use this information to act in the role of the client against other servers. For a client that accesses a NaturalX server, a similar consideration to that inthe section Server Configuration - General Settings applies: only if it specifies at least *Identify*, will the NaturalX server be able to retrieve its user ID and use it against Natural Security. If this entry is defined differently on the client side and on the server side, the stricter setting applies.

Client Configuration - Application-Specific Settings

The application-specific settings can be set up differently for each NaturalX application. But the question is where to apply these settings. Remember that all classes registered under one NaturalX server ID form one application in the DCOM sense, and are thus assigned to one AppID key in the registry. This is why the application-specific settings are applied under the AppID key.

- The registry key *HKCR\AppID\<APPID>\RemoteServerName* defines on which remote machine DCOM should start the server when a class hosted by this server is requested. If the server is to be started locally, ‘Run on this computer’ and no *RemoteServerName* must be specified.

Sample Application

On Windows 98/NT/2000, a sample application is provided in the library SYSEXNXX. For information on how to run this application, see the A-README member in the library SYSEXNXX.

